**User , Login , Permission , And other Concepts**

1. **User & Login**

* **User & login Created To Secure Database**
* **User has Permission to Work in Database in Which User is Create.**
* **But, WithOut permission of Owener. User can not DML, DDL in Database**
* **Accept his Own Database 🡪 He Can not See other database**
* **1 database kaa Multiple User bnaaa sktaa han**
* **Users has there Login** 
  1. **Create**

use practice1 --> User will be created in this Database

--\_\_ Create \_\_

Create Login saqib1 with Passwor = 'saqib123'

Create user saqib for login saqib1

* 1. **Alter**

---\_\_ Alter \_\_

alter login saqib1 with password = 'saqib111';

* 1. **Drop**

--\_\_ Drop \_\_\_

DROP USER user\_name;

1. **List of User, Logins, Permissions of User**
   1. **Users List of Selected Database**

use practice1 –-of selectd database

--List of User of selectd database

SELECT name

FROM sys.database\_principals

WHERE type\_desc = 'SQL\_USER';

* 1. **List of Login**

SELECT name

FROM sys.server\_principals

WHERE type\_desc = 'SQL\_LOGIN';

* 1. **List of Permission**

--\_\_\_\_\_\_\_ 1st Way \_\_\_\_\_\_\_\_

EXEC sp\_helprotect @username = 'Saqib';

--\_\_\_\_\_\_\_ 2nd Way \_\_\_\_\_\_\_

SELECT \*

FROM sys.database\_permissions

WHERE grantee\_principal\_id = DATABASE\_PRINCIPAL\_ID('saqib');

1. ***Permission Levels***
   1. **Database-level Permission**
2. this type of permission is granted at the database level and applies to all objects within that database. Examples include:

* CONNECT: Allows a user to connect to the database.
* CREATE PROCEDURE: Allows a user to create stored procedures in the database.
* BACKUP DATABASE: Allows a user to create backups of the database.
  + CONTROL - Allows a user to perform any operation on a database.
  + ALTER ANY DATABASE - Allows a user to alter any database on the server.
  + CREATE DATABASE - Allows a user to create a new database on the server.
  + VIEW ANY DATABASE - Allows a user to see all databases on the server.
  + IMPERSONATE ANY LOGIN - Allows a user to impersonate any login on the server.
  + BACKUP DATABASE - Allows a user to back up any database on the server.
  + RESTORE DATABASE - Allows a user to restore any database on the server.
  + TAKE OWNERSHIP - Allows a user to take ownership of any securable object on the server.
  + CONTROL SERVER - Allows a user to perform any operation on the server.
  + SHUTDOWN - Allows a user to shut down the server.

--\_\_\_\_\_\_\_\_\_\_\_\_ 1. database level permission \_\_\_\_\_\_\_\_\_\_\_\_

--\_\_ database connect

revoke connect to saqib;

--check selected database after login -- not work

--\_\_\_\_ schema

GRANT CREATE schema TO saqib;

--\_\_ Permission daana

grant connect to saqib;

* 1. **Schema-level permission**

1. This type of permission is granted at the schema level and applies to all objects within that schema. Examples include:

* CREATE TABLE: Allows a user to create tables within the schema.
* CREATE VIEW: Allows a user to create views within the schema.
* CREATE FUNCTION: Allows a user to create user-defined functions within the schema.

--\_\_\_\_\_\_\_\_\_\_\_\_ 2. Schema-level permission \_\_\_\_\_\_\_\_\_\_\_\_

--by default user ka pass nhin hotaa

GRANT CREATE TABLE TO saqib;

revoke create table to saqib

--- after thes ---> database level permission of ----> Schema dbo.

GRANT SELECT ON SCHEMA::b TO saqib; --database level permission

* 1. **Object Level Permission (table)**

1. This type of permission is granted at the object level (such as a table, view, or stored procedure) and applies only to that object. Object-level permissions control who can read, write, or execute specific objects within the database. Examples of object-level permissions include:

* SELECT: Allows a user to read data from the object.
* INSERT: Allows a user to insert new data into the object.
* EXECUTE: Allows a user to execute the object (such as a stored procedure or user-defined function).

grant SELECT ON customer TO saqib;

revoke SELECT ON customer TO saqib;

* 1. **Column Level permission**

1. This type of permission is granted at the column level within a table or view, and applies only to that column. Column-level permissions control who can read or write specific columns within a table or view. Examples of column-level permissions include:

* SELECT: Allows a user to read data from the column.
* UPDATE: Allows a user to modify data in the column.

GRANT SELECT (column\_name) ON table\_name TO user\_name;

revoke select (column\_name) ON table\_name TO user\_name;

1. **Roles :** Roles are used to group users and permissions together, making it easier to manage permissions across multiple users. There are two types of roles in SQL Server: fixed roles and user-defined roles. Fixed roles are predefined and have specific permissions associated with them, while user-defined roles are created by users and can have custom permissions.

To create a user-defined role, use the following syntax:

sqlCopy code

CREATE ROLE role\_name;

To add a user to a role, use the following syntax:

sqlCopy code

EXEC sp\_addrolemember 'role\_name', 'user\_name';

To grant permissions to a role, use the same syntax as granting permissions to a user, but replace the user name with the role name.

1. Securables: Securables are the objects that permissions can be granted on, such as tables, views, and stored procedures. There are several types of securables in SQL Server, including database-level securables and server-level securables.

To view the securables in a database, use the following syntax:

sqlCopy code

SELECT \* FROM sys.database\_permissions;

To view the server-level securables, use the following syntax:

sqlCopy code

SELECT \* FROM sys.server\_permissions;

1. Ownership Chaining: Ownership chaining is a mechanism in SQL Server that allows users to access objects in a database without needing explicit permissions on those objects. It works by allowing a user to access an object if they have permissions on the object's parent, and the owner of the object is the same as the owner of the parent.

To enable ownership chaining, use the following syntax:

sqlCopy code

ALTER DATABASE database\_name SET DB\_CHAINING ON;

1. Cross-Database Ownership Chaining: Cross-database ownership chaining is similar to ownership chaining, but it allows a user to access objects in another database if they have permissions on the object's parent, and the owner of the object and the owner of the parent are the same in both databases.

To enable cross-database ownership chaining, use the following syntax:

sqlCopy code

EXEC sp\_configure 'cross db ownership chaining', 1; RECONFIGURE;

1. Impersonation: Impersonation is a feature in SQL Server that allows a user to execute a stored procedure or function using the permissions of another user. This can be useful in scenarios where a user needs to access objects that they don't have permissions on directly.

To enable impersonation, use the following syntax:

vbnetCopy code

GRANT IMPERSONATE ON user\_name TO user2\_name;